Performance Optimization of N-Body Simulations Using Serial, OpenMP, and Kokkos Approaches

Ahmed Eltayeb
Department of Computer Science
University of Salerno
Italy

***Abstract*—This study evaluates N-body simulations implemented using serial execution, OpenMP, and Kokkos frameworks on CPU and GPU architectures. Simulations involve 10,000 particles over 100 time steps using a naive all pairs interaction model. The serial implementation establishes a baseline with an average runtime of 78.5 seconds. OpenMP and Kokkos CPU implementations achieve speedups of 8.7x and 11.7x, respectively, through multithreading. GPU implementations using Kokkos with SYCL show performance variation depending on precision: double-precision achieves 3.1x speedup, while single-precision reaches 8.8x. All implementations maintain numerical consistency, with deviations limited to floating point precision and execution order. Performance differences are attributed to parallelization strategy, memory layout, and hardware. The results provide quantitative insight into the efficiency of portable parallel programming models.**

I. Introduction

N-body simulations model the dynamics of systems with pairwise interactions, such as gravitational or electrostatic forces. They have applications in fields like physics, astronomy, and engineering. However, the computational complexity of O(N²) makes them intensive as particle numbers grow.

This project optimizes an N-body simulation by comparing three implementations: a serial version, a parallel OpenMP version, and a portable Kokkos version. Each represents a step in the optimization process, from a basic sequential algorithm to scalable parallel techniques.

The serial version serves as a baseline. OpenMP adds CPU parallelism with minimal code restructuring. Kokkos provides a higher level, architecture agnostic approach that supports performance portability across different hardware backends, including multicore CPUs and GPUs.

This report evaluates the implementations based on execution time, parallelization strategies, memory access patterns, and scalability, aiming to understand the tradeoffs between ease of implementation, portability, and performance.

Motivation. The N-body problem is fundamental in computational physics and astrophysics, modeling systems under gravitational attraction. Simulating these systems is compute intensive, with quadratic complexity. Performance optimization is crucial for large datasets or real time applications.

Optimizing N-body simulations is challenging due to nonuniform memory access patterns, synchronization, and efficient hardware utilization. This report implements and compares different versions of the N-body simulation: serial, OpenMP, Kokkos CPU and Kokkos GPU. The goal is to demonstrate performance improvements and highlight the trade offs in complexity and scalability.

Related Work. The N-body problem has been widely studied, with early work by Aarseth (1963) introducing direct summation methods, which are still used for their simplicity and accuracy. Barnes and Hut (1986) proposed a hierarchical tree algorithm that reduces complexity to $O(N \log N)$, while Greengard and Rokhlin (1987) developed the Fast Multipole Method (FMM), improving scalability.

Parallel computing has been explored for N-body simulations. OpenMP offers an easy way to parallelize the problem on shared memory systems (Rahman et al., 2017), while MPI (Message Passing Interface) scales across distributed systems but requires complex data exchange (Dubinski, 1996). Recently, performance portability libraries like Kokkos (Edwards et al., 2014) abstract hardware details for efficient parallelism on CPUs and GPUs.

This study focuses on performance optimization, comparing different implementations: naive serial, OpenMP, Kokkos CPU and Kokkos Kokkos GPU, without developing new algorithms.

II. Background: The N-Body Simulation Problem

The N-body Problem. The N-body problem predicts the motion of N particles under mutual gravitational influence. Each particle has a position vector $\vec{r}_i$, velocity vector $\vec{v}_i$, and mass m_i. The gravitational force between particles i and j is:

$$\vec{F}_{ij} = G\frac{m_i m_j}{\|\vec{r}_j - \vec{r}_i\|^3}(\vec{r}_j - \vec{r}_i)$$

The total force on each particle is computed as the sum of interactions with all others, and Newton's second law is used to update its velocity and position.

Serial Algorithm. The serial algorithm calculates pairwise forces for each particle, resulting in $O(N^2)$ force evaluations per time step. Position and velocity updates are done using Euler integration:

$$\vec{v}_i(t+\Delta t) = \vec{v}_i(t) + \frac{\vec{F}_i}{m_i}\Delta t, \quad \vec{r}_i(t+\Delta t) = \vec{r}_i(t) + \vec{v}_i(t+\Delta t)\Delta t$$

Parallel Execution Models. The problem is highly parallelizable, with independent force calculations for each particle. Two parallel programming models are Explored:

- **OpenMP**: Shared memory multithreading on CPUs.

- **Kokkos**: A performance portability framework that abstracts hardware parallelism.

Summary. This background outlines the mathematical model of the N-body problem, the serial algorithm used to compute particle trajectories, and the parallel programming models that form the basis of our optimizations.

III. Proposed Method

The project implements three distinct versions of the N-body simulation in order to compare performance. All versions simulate the gravitational interactions of 10,000 particles over 100 time steps.

A. Serial Version

This implementation serves as the performance baseline. It is a straightforward algorithm that uses a single thread and computes the force on each particle by iterating through every other particle. The force calculation involves a nested loop, resulting in a computational complexity of $O(N^2)$, which is highly inefficient for large numbers of particles.

B. OpenMP Version

This implementation introduces shared memory parallelism. It uses compiler pragmas to parallelize the main force calculation loop. The directive `#pragma omp parallel for` is applied to the outer loop. This allows the workload to be distributed among multiple CPU cores, reducing the overall execution time.

C. Kokkos Versions

Kokkos is a C++ performance portability framework designed to abstract parallel execution and data layout. Kokkos was used to implement a CPU based version as well as two GPU based versions. All Kokkos versions utilize a Structure of Arrays (SoA) memory layout with `Kokkos::View`, which is an array abstraction that manages memory on different devices (CPU, GPU, etc.) and provides performance portability across backends such as SYCL, CUDA, and OpenMP.

The CPU version uses the OpenMP backend of Kokkos to execute `parallel_for` kernels on the CPU cores. The GPU versions use the SYCL backend to target an Intel integrated GPU. Two GPU versions were implemented to compare the performance of double-precision and single-precision data types, as single-precision can offer significant speed advantages on GPUs.

IV. Experimental Results

In this section, the performance and the correctness of different N-body implementations is evaluated, the runtime is measured across multiple parallelization strategies and the impact of different Kokkos execution models is analyzed.

Experimental setup. The experiments were conducted on a machine with an Intel Core i5 Processor and an integrated Intel UHD Graphics card, running Windows 11. The Kokkos GPU benchmarks were executed within a Windows Subsystem for Linux (WSL) environment to leverage Intel oneAPI tools. The CPU implementations of the code were compiled using `icx`, the Intel oneAPI DPC++/C++ compiler optimized for high performance CPU execution. For the GPU implementations, the `icpx` compiler was used, which extends `icx` with SYCL (Data Parallel C++) support to target heterogeneous hardware such as GPUs and other accelerators. Both compilers are part of the Intel oneAPI toolkit, designed to provide performance portability across different architectures.

The OpenMP and Kokkos CPU executions were configured to use 12 threads to fully utilize the available CPU cores by setting the environment variable `OMP_NUM_THREADS=12`.

The benchmark consists of simulating the gravitational interactions between 10000 particles over 100 discrete time steps using a naive all pairs interaction model ($O(N^2)$). All implementations use identical randomized initial conditions to ensure result consistency and fair comparisons. Each program prints the positions of the first four particles after the simulation to verify that different versions produce numerically similar outcomes.

Results. The following versions were benchmarked:

- **Serial**: A single threaded implementation using basic C++ loops and STL vectors.
- **OpenMP**: A parallelized version using `#pragma omp parallel for` to distribute work across CPU threads.
- **Kokkos CPU**: A version using the OpenMP backend of Kokkos on the CPU.
- **Kokkos GPU (double)**: A GPU version using the SYCL backend with double precision floating point types.
- **Kokkos GPU (float)**: A GPU version using the SYCL backend with single precision floating point types.

Runtime comparison: The table below shows the total runtime of each implementation.

Implementation	Average runtime (s)
Serial	78.489
OpenMP	9.036
Kokkos CPU	6.711
Kokkos GPU (double)	25.052
Kokkos GPU (float)	8.928

The serial version serves as a baseline. The OpenMP implementation achieves a significant speedup of approximately $8.7\times$ over the serial version. The Kokkos CPU version performs better achieving a speedup of approximately $11.7\times$. This demonstrates that for this specific hardware configuration with an integrated Intel GPU, the Kokkos CPU implementation is the solution that performs the best.

The GPU implementations provided a comparison of floating point precision. The Kokkos GPU version using double-precision types was slower than the CPU versions. However, when switching to single-precision floating point types, the Kokkos GPU version achieved a speedup of approximately $8.8\times$ over the serial version, nearly matching the performance of the OpenMP implementation.

Numerical consistency: All versions using the same initial conditions (OpenMP, serial, Kokkos CPU) produced nearly identical particle positions at the end of the simulation. Versions using the same layout and integration logic (e.g., double vs. float) showed small expected differences due to floating point precision and execution order, but no divergence. This confirms that all implementations compute functionally correct and consistent results.

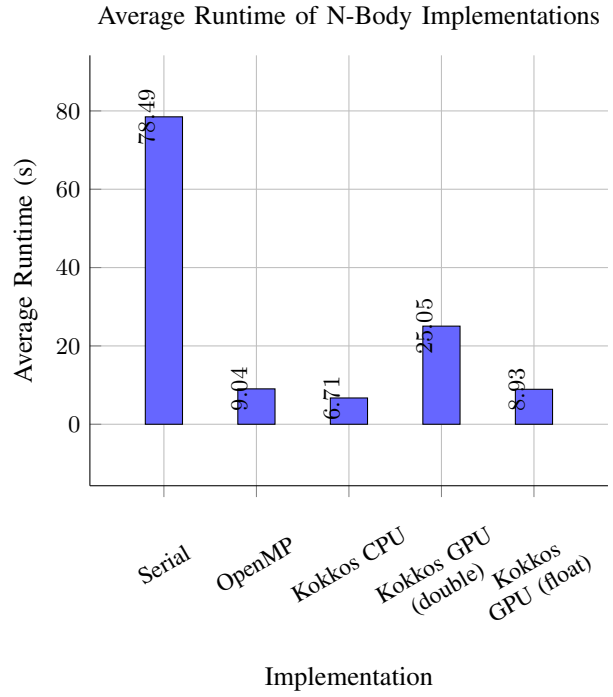

Fig. 1. Average runtime comparison of the different N-body simulation implementations for 10,000 particles.

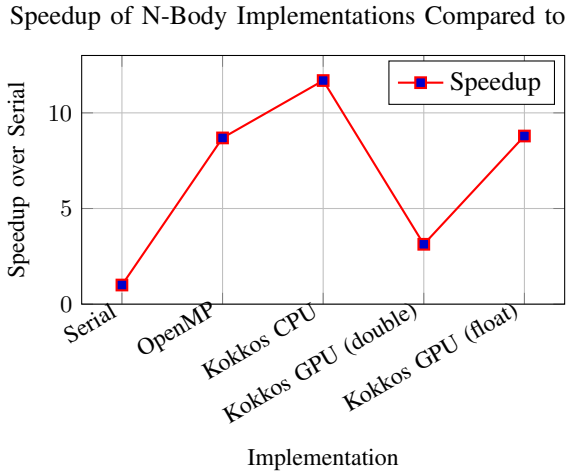

Fig. 2. Speedup of parallel N-body implementations relative to the serial baseline.

V. Conclusions

This work implemented and evaluated N-body simulations using serial, OpenMP, and Kokkos approaches on CPU and GPU architectures. Execution times were measured for 10,000 particles over 100 time steps, revealing that multithreaded CPU implementations (OpenMP and Kokkos CPU) achieved substantial speedups over the serial baseline, while GPU performance depended strongly on floating point precision. Single-precision Kokkos GPU executions approached the performance of CPU parallel versions, whereas double-precision executions were slower due to hardware constraints. All implementations produced numerically consistent results within expected floating point tolerances. These findings demonstrate that Kokkos provides a portable programming model capable of leveraging heterogeneous hardware efficiently without modifying the core algorithm. The study highlights the impact of memory layout, parallelization strategy, and precision on performance, providing a quantitative basis for selecting appropriate implementations for compute intensive simulations. Future work could explore hierarchical algorithms, larger particle counts, and alternative accelerator hardware to further assess scalability and efficiency.

References

[1] S. J. Aarseth, *Dynamical evolution of clusters of galaxies, I*, Monthly Notices of the Royal Astronomical Society, vol. 126, no. 3, pp. 223–255, 1963.

[2] J. Barnes and P. Hut, *A hierarchical O(N log N) force-calculation algorithm*, Nature, vol. 324, pp. 446–449, 1986.

[3] L. Greengard and V. Rokhlin, *A fast algorithm for particle simulations*, Journal of Computational Physics, vol. 73, no. 2, pp. 325–348, 1987.

[4] M. M. Rahman, F. Ahmed, and M. A. Rahman, *Parallelizing N-body simulation using OpenMP*, 2017 International Conference on Electrical, Computer and Communication Engineering (ECCE), pp. 269–273, 2017.

[5] J. Dubinski, *A parallel tree code*, New Astronomy, vol. 1, no. 2, pp. 133–147, 1996.

[6] H. C. Edwards, C. R. Trott, and D. Sunderland, *Kokkos: Enabling manycore performance portability through polymorphic memory access patterns*, Journal of Parallel and Distributed Computing, vol. 74, no. 12, pp. 3202–3216, 2014.